PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 9/30	A1	(11) International Publication Number: WO 93/01543 (43) International Publication Date: 21 January 1993 (21.01.93)

(21) International Application Number: PCT/US92/05720

(22) International Filing Date: 8 July 1992 (08.07.92)

(30) Priority data:
726,773 8 July 1991 (08.07.91) US

(71) Applicant: S-MOS SYSTEMS, INC. [US/US]; 2460 North First Street, San Jose, CA 95131-1002 (US).

(72) Inventors: GARG, Sanjiv ; 46820 Sentinel Drive, Fremont, CA 94539 (US). LENTZ, Derek, J. ; 17400 Phillips Avenue, Los Gatos, CA 95032 (US). NGUYEN, Le Trong ; 15096 Danielle Place, Monte Sereno, CA 95030 (US). CHEN, Sho, Long ; 14411 Quito Road, Saratoga, CA 95070 (US).

(74) Agents: KESSLER, Edward, J. et al.; Sterne, Kessler, Goldstein & Fox, 1225 Connecticut Avenue, N.W., Suite 300, Washington, DC 20036 (US).

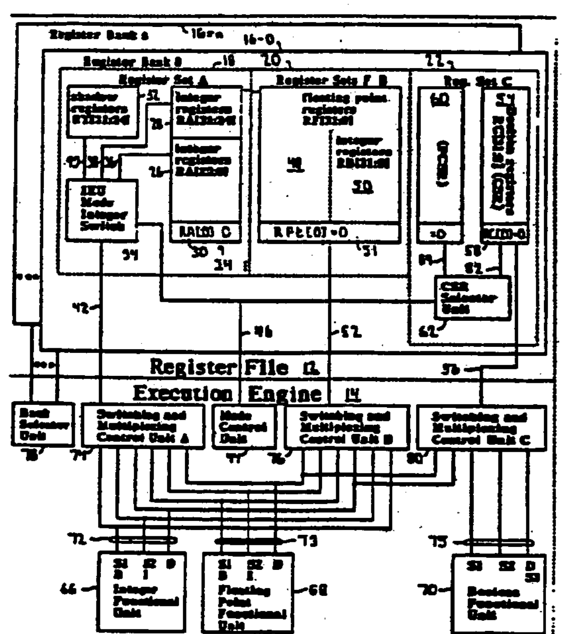
(81) Designated States: JP, KR, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IT, LU, MC, NL, SE).

Published
With international search report.

(54) Title: RISC MICROPROCESSOR ARCHITECTURE IMPLEMENTING MULTIPLE TYPED REGISTER SETS

(57) Abstract

A register system for a data processor which operates in a plurality of modes. The register system provides multiple, identical banks of register sets, the data processor controlling access such that instructions and processes need not specify any given bank. An integer register set includes first (RA[23:0]) and second (RA[31:24]) subsets, and a shadow subset (RT[31:24]). While the data processor is in a first mode, instructions access the first and second subsets. While the data processor is in a second mode, instructions may access the first subset, but any attempts to access the second subset are re-routed to the shadow subset instead, transparently to the instructions, allowing system routines to seemingly use the second subset without having to save and restore data which user routines have written to the second subset. A re-typable register set provides integer whidth data and floating point width data in response to integer instructions and floating point instructions, respectively. Boolean comparison instructions specify particular integer or floating point registers for source data to be compared, and specify a particular Boolean register for the result, so there are no dedicated, fixed-location status flags. Boolean combinational instructions combine specified Boolean registers for performing complex Boolean comparisons without intervening conditional branch instructions, to minimize pipeline disruption.



10 Instruction Execution Unit (IEU)

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FI	Finland	ML	Mali
AU	Australia	FR	France	MN	Mongolia
BB	Barbados	GA	Gabon	MR	Mauritania
BE	Belgium	GB	United Kingdom	MW	Malawi
BF	Burkina Faso	GN	Guinea	NL	Netherlands
BG	Bulgaria	GR	Greece	NO	Norway
BJ	Benin	HU	Hungary	PL	Poland
BR	Brazil	IE	Ireland	RO	Romania
CA	Canada	IT	Italy	RU	Russian Federation
CF	Central African Republic	JP	Japan	SD	Sudan
CG	Congo	KP	Democratic People's Republic of Korea	SE	Sweden
CH	Switzerland			SN	Senegal
CI	Côte d'Ivoire	KR	Republic of Korea	SU	Soviet Union
CM	Cameroon	LI	Liechtenstein	TD	Chad
CS	Czechoslovakia	LK	Sri Lanka	TG	Togo
DE	Germany	LU	Luxembourg	US	United States of America
DK	Denmark	MC	Monaco		
ES	Spain	MG	Madagascar		

RISC MICROPROCESSOR ARCHITECTURE IMPLEMENTING MULTIPLE TYPED REGISTER SETS

CROSS-REFERENCE TO RELATED APPLICATIONS

Applications of particular interest to the present application, include:

1. HIGH-PERFORMANCE RISC MICROPROCESSOR ARCHITECTURE, SC/Serial No. 07/727,006, filed 08 July 1991 by Le T. Nguyen et al;

2. EXTENSIBLE RISC MICROPROCESSOR ARCHITECTURE, SC/Serial No. 07/727,058, filed 08 July 1991 by Le T. Nguyen et al;

3. RISC MICROPROCESSOR ARCHITECTURE WITH ISOLATED ARCHITECTURAL DEPENDENCIES, SC/Serial No. 07/726,744, filed 08 July 1991 by Le T. Nguyen et al;

4. RISC MICROPROCESSOR ARCHITECTURE IMPLEMENTING FAST TRAP AND EXCEPTION STATE, SC/Serial No. 07/726,942, filed 08 July 1991 by Le T. Nguyen et al;

5. SINGLE CHIP PAGE PRINTER CONTROLLER, SC/Serial No. 07/726,929, filed 08 July 1991 by Derek J. Lentz et al;

6. MICROPROCESSOR ARCHITECTURE CAPABLE OF SUPPORTING MULTIPLE HETEROGENEOUS PROCESSORS, SC/Serial No. 07/726,893, filed 08 July 1991 by Derek J. Lentz et al.

The above-identified Applications are hereby incorporated herein by reference, their collective teachings being part of the present disclosure.

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to microprocessors, and more specifically to a RISC microprocessor having plural, symmetrical sets of registers.

Description of the Background

In addition to the usual complement of main memory storage and secondary permanent storage, a microprocessor-based computer system typically also includes one or more general purpose data registers, one or more address registers, and one or more status flags. Previous systems have included integer registers for

holding integer data and floating point registers for holding floating point data. Typically, the status flags are used for indicating certain conditions resulting from the most recently executed operation. There generally are status flags for indicating whether, in the previous operation: a carry occurred, a negative number resulted, and/or a zero resulted.

These flags prove useful in determining the outcome of conditional branching within the flow of program control. For example, if it is desired to compare a first number to a second number and upon the conditions that the two are equal, to branch to a given subroutine, the microprocessor may compare the two numbers by subtracting one from the other, and setting or clearing the appropriate condition flags. The numerical value of the result of the subtraction need not be stored. A conditional branch instruction may then be executed, conditioned upon the status of the zero flag. While being simple to implement, this scheme lacks flexibility and power. Once the comparison has been performed, no further numerical or other operations may be performed before the conditional branch upon the appropriate flag; otherwise, the intervening instructions will overwrite the condition flag values resulting from the comparison, likely causing erroneous branching. The scheme is further complicated by the fact that it may be desirable to form greatly complex tests for branching, rather than the simple equality example given above.

For example, assume that the program should branch to the subroutine only upon the condition that a first number is greater

than a second number, and a third number is less than a fourth number, and a fifth number is equal to a sixth number. It would be necessary for previous microprocessors to perform a lengthy series of comparisons heavily interspersed with conditional branches. A particularly undesirable feature of this serial scheme of comparing and branching is observed in any microprocessor having an instruction pipeline.

In a pipelined microprocessor, more than one instruction is being executed at any given time, with the plural instructions being in different stages of execution at any given moment. This provides for vastly improved throughput. A typical pipeline microprocessor may include pipeline stages for: (a) fetching an instruction, (b) decoding the instruction, (c) obtaining the instruction's operands, (d) executing the instruction, and (e) storing the results. The problem arises when a conditional branch instruction is fetched. It may be the case that the conditional branch's condition cannot yet be tested, as the operands may not yet be calculated, if they are to result from operations which are yet in the pipeline. This results in a "pipeline stall", which dramatically slows down the processor.

Another shortcoming of previous microprocessor-based systems is that they have included only a single set of registers of any given data type. In previous architectures, when an increased number of registers has been desired within a given data type, the solution has been simply to increase the size of the single set of those type of registers. This may result in addressing problems, access conflict problems, and symmetry problems.

On a similar note, previous architectures have restricted each given register set to one respective numerical data type. Various prior systems have allowed general purpose registers to hold either numerical data or address "data", but the present application will not use the term "data" to include addresses. What is intended may be best understood with reference to two prior systems. The Intel 8085 microprocessor includes a register pair "HL" which can be used to hold either two bytes of numerical data or one two-byte address. The present application's improvement is not directed to that issue. More on point, the Intel 80486 microprocessor includes a set of general purpose integer data registers and a set of floating point registers, with each set being limited to its respective data type, at least for purposes of direct register usage by arithmetic and logic units.

This proves wasteful of the microprocessor's resources, such as the available silicon area, when the microprocessor is performing operations which do not involve both data types. For example, user applications frequently involve exclusively integer operations, and perform no floating point operations whatsoever. When such a user application is run on a previous microprocessor which includes floating point registers (such as the 80486), those floating point registers remain idle during the entire execution.

Another problem with previous microprocessor register set architecture is observed in context switching or state switching between a user application and a higher access privilege level

entity such as the operating system kernel. When control within the microprocessor switches context, mode, or state, the operating system kernel or other entity to which control is passed typically does not operate on the same data which the user application has been operating on. Thus, the data registers typically hold data values which are not useful to the new control entity but which must be maintained until the user application is resumed. The kernel must generally have registers for its own use, but typically has no way of knowing which registers are presently in use by the user application. In order to make space for its own data, the kernel must swap out or otherwise store the contents of a predetermined subset of the registers. This results in considerable loss of processing time to overhead, especially if the kernel makes repeated, short-duration assertions of control.

On a related note, in prior microprocessors, when it is required that a "grand scale" context switch be made, it has been necessary for the microprocessor to expend even greater amounts of processing resources, including a generally large number of processing cycles, to save all data and state information before making the switch. When context is switched back, the same performance penalty has previously been paid, to restore the system to its former state. For example, if a microprocessor is executing two user applications, each of which requires the full complement of registers of each data type, and each of which may be in various stages of condition code setting operations or numerical calculations, each switch from one user

application to the other necessarily involves swapping or otherwise saving the contents of every data register and state flag in the system. This obviously involves a great deal of operational overhead, resulting in significant performance degradation, particularly if the main or the secondary storage to which the registers must be saved is significantly slower than the microprocessor itself.

Therefore, we have discovered that it is desirable to have an improved microprocessor architecture which allows the various component conditions of a complex condition to be calculated without any intervening conditional branches. We have further discovered that it is desirable that the plural simple conditions be calculable in parallel, to improve throughput of the microprocessor.

We have also discovered that it is desirable to have an architecture which allows multiple register sets within a given data type.

Additionally, we have discovered it to be desirable for a microprocessor's floating point registers to be usable as integer registers, in case the available integer registers are inadequate to optimally to hold the necessary amount of integer data. Notably, we have discovered that it is desirable that such re-typing be completely transparent to the user application.

We have discovered it to be highly desirable to have a microprocessor which provides a dedicated subset of registers which are reserved for use by the kernel in lieu of at least a subset of the user registers, and that this new set of registers

should be addressable in exactly the same manner as the register subset which they replace, in order that the kernel may use the same register addressing scheme as user applications. We have further observed that it is desirable that the switch between the two subsets of registers require no microprocessor overhead cycles, in order to maximally utilize the microprocessor's resources.

Also, we have discovered it to be desirable to have a microprocessor architecture which allows for a "grand scale" context switch to be performed with minimal overhead. In this vein, we have discovered that is desirable to have an architecture which allows for plural banks of register sets of each type, such that two or more user applications may be operating in a multi-tasking environment, or other "simultaneous" mode, with each user application having sole access to at least a full bank of registers. It is our discovery that the register addressing scheme should, desirably, not differ between user applications, nor between register banks, to maximize simplicity of the user applications, and that the system should provide hardware support for switching between the register banks so that the user applications need not be aware of which register bank which they are presently using or even of the existance of other register banks or of other user applications.

These and other advantages of our invention will be appreciated with reference to the following description of our invention, the accompanying drawings, and the claims.

SUMMARY OF THE INVENTION

The present invention provides a register file system comprising: an integer register set including first and second subsets of integer registers, and a shadow subset; a re-typable set of registers which are individually usable as integer registers or as floating point registers; and a set of individually addressable Boolean registers.

The present invention includes integer and floating point functional units which execute integer instructions accessing the integer register set, and which operate in a plurality of modes. In any mode, instructions are granted ordinary access to the first subset of integer registers. In a first mode, instructions are also granted ordinary access to the second subset. However, in a second mode, instructions attempting to access the second subset are instead granted access to the shadow subset, in a manner which is transparent to the instructions. Thus, routines may be written without regard to which mode they will operate in, and system routines (which operate in the second mode) can have at least the second subset seemingly at their disposal, without having to expend the otherwise-required overhead of saving the second subset's contents (which may be in use by user processes operating in the first mode).

The invention further includes a plurality of integer register sets, which are individually addressable as specified by fields in instructions. The register sets include read ports and write ports which are accessed by multiplexers, wherein the

multiplexers are controlled by contents of the register set-specifying fields in the instructions.

One of the integer register sets is also usable as a floating point register set. In one embodiment, this set is sixty-four bits wide to hold double-precision floating point data, but only the low order thirty-two bits are used by integer instructions.

The invention includes functional units for performing Boolean operations, and further includes a Boolean register set for holding results of the Boolean operations such that no dedicated, fixed-location status flags are required. The integer and floating point functional units execute numerical comparison instructions, which specify individual ones of the Boolean registers to hold results of the comparisons. A Boolean functional unit executes Boolean combinational instructions whose sources and destination are specified registers in the Boolean register set. Thus, the present invention may perform conditional branches upon a single result of a complex Boolean function without intervening conditional branch instructions between the fundamental parts of the complex Boolean function, minimizing pipeline disruption in the data processor.

Finally, there are multiple, identical register banks in the system, each bank including the above-described register sets. A bank may be allocated to a given process or routine, such that the instructions within the routine need not specify upon which bank they operate.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram of the instruction execution unit of the microprocessor of the present invention, showing the elements of the register file.

Figs. 2-4 are simplified schematic and block diagrams of the floating point, integer and Boolean portions of the instruction execution unit of Fig. 1, respectively.

Figs. 5-6 are more detailed views of the floating point and integer portions, respectively, showing the means for selecting between register sets.

Fig. 7 illustrates the fields of an exemplary microprocessor instruction word executable by the instruction execution unit of Fig. 1.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

I. REGISTER FILE

Fig. 1 illustrates the basic components of the instruction execution unit (IEU) 10 of the RISC (reduced instruction set computing) processor of the present invention. The IEU 10 includes a register file 12 and an execution engine 14. The register file 12 includes one or more register banks 16-0 to 16-n. It will be understood that the structure of each register bank 16 is identical to all of the other register banks 16. Therefore, the present application will describe only register bank 16-0. The register bank includes a register set A 18, a register set FB 20, and a register set C 22.

In general, the invention may be characterized as a RISC microprocessor having a register file optimally configured for use in the execution of RISC instructions, as opposed to conventional register files which are sufficient for use in the execution of CISC (complex instruction set computing) instructions by CISC processors. By having a specially adapted register file, the execution engine of the microprocessor's IEU achieves greatly improved performance, both in terms of resource utilization and in terms of raw throughput. The general concept is to tune a register set to a RISC instruction, while the specific implementation may involve any of the register sets in the architecture.

A. Register Set A

Register set A 18 includes integer registers 24 (RA[31:0]), each of which is adapted to hold an integer value datum. In one embodiment, each integer may be thirty-two bits wide. The RA[] integer registers 24 include a first plurality 26 of integer registers (RA[23:0]) and a second plurality 28 of integer registers (RA[31:24]). The RA[] integer registers 24 are each of identical structure, and are each addressable in the same manner, albeit with a unique address within the integer register set 24. For example, a first integer register 30 (RA[0]) is addressable at a zero offset within the integer register set 24.

RA[0] always contains the value zero. It has been observed that user applications and other programs use the constant value zero more than any other constant value. It is, therefore,

desirable to have a zero readily available at all times, for clearing, comparing, and other purposes. Another advantage of having a constant, hard-wired value in a given register, regardless of the particular value, is that the given register may be used as the destination of any instruction whose results need not be saved.

Also, this means that the fixed register will never be the cause of a data dependency delay. A data dependency exists when a "slave" instruction requires, for one or more of its operands, the result of a "master" instruction. In a pipelined processor, this may cause pipeline stalls. For example, the master instruction, although occurring earlier in the code sequence than the slave instruction, may take considerably longer to execute. It will be readily appreciated that if a slave "increment and store" instruction operates on the result data of a master "quadruple-word integer divide" instruction, the slave instruction will be fetched, decoded, and awaiting execution many clock cycles before the master instruction has finished execution. However, in certain instances, the numerical result of a master instruction is not needed, and the master instruction is executed for some other purpose only, such as to set condition code flags. If the master instruction's destination is RA[0], the numerical results will be effectively discarded. The data dependency checker (not shown) of the IEU 10 will not cause the slave instruction to be delayed, as the ultimate result of the master instruction -- zero -- is already known.

The integer register set A 24 also includes a set of shadow registers 32 (RT[31:24]). Each shadow register can hold an integer value, and is, in one embodiment, also thirty-two bits wide. Each shadow register is addressable as an offset in the same manner in which each integer register is addressable.

Finally, the register set A includes an IEU mode integer switch 34. The switch 34, like other such elements, need not have a physical embodiment as a switch, so long as the corresponding logical functionality is provided within the register sets. The IEU mode integer switch 34 is coupled to the first subset 26 of integer registers on line 36, to the second subset of integer registers 28 on line 38, and to the shadow registers 32 on line 40. All accesses to the register set A 18 are made through the IEU mode integer switch 34 on line 42. Any access request to read or write a register in the first subset RA[23:0] is passed automatically through the IEU mode integer switch 34. However, accesses to an integer register with an offset outside the first subset RA[23:0] will be directed either to the second subset RA[31:24] or the shadow registers RT[31;24], depending upon the operational mode of the execution engine 14.

The IEU mode integer switch 34 is responsive to a mode control unit 44 in the execution engine 14. The mode control unit 44 provides pertinent state or mode information about the IEU 10 to the IEU mode integer switch 34 on line 46. When the execution engine performs a context switch such as a transfer to kernel mode, the mode control unit 44 controls the IEU mode integer switch 34 such that any requests to the second subset

RA[31:24] are re-directed to the shadow RT[31:24], using the same requested offset within the integer set. Any operating system kernel or other then-executing entity may thus have *apparent* access to the second subset RA[31:24] without the otherwise-required overhead of swapping the contents of the second subset RA[31:24] out to main memory, or pushing the second subset RA[31:24] onto a stack, or other conventional register-saving technique.

When the execution engine 14 returns to normal user mode and control passes to the originally-executing user application, the mode control unit 44 controls the IEU mode integer switch 34 such that access is again directed to the second subset RA[31:24]. In one embodiment, the mode control unit 44 is responsive to the present state of interrupt enablement in the IEU 10. In one embodiment, the execution engine 14 includes a processor status register (PSR) (not shown), which includes a one-bit flag (PSR[7]) indicating whether interrupts are enabled or disabled. Thus, the line 46 may simply couple the IEU mode integer switch 34 to the interrupts-enabled flag in the PSR. While interrupts are disabled, the IEU 10 maintains access to the integers RA[23:0], in order that it may readily perform analysis of various data of the user application. This may allow improved debugging, error reporting, or system performance analysis.

B. <u>Register Set FB</u>

The re-typable register set FB 20 may be thought of as including floating point registers 48 (RF[31:0]); and/or integer

registers 50 (RB[31:0]). When neither data type is implied to the exclusion of the other, this application will use the term RFB[]. In one embodiment, the floating point registers RF[] occupy the same physical silicon space as the integer registers RB[]. In one embodiment, the floating point registers RF[] are sixty-four bits wide and the integer registers RB[] are thirty-two bits wide. It will be understood that if double-precision floating point numbers are not required, the register set RFB[] may advantageously be constructed in a thirty-two-bit width to save the silicon area otherwise required by the extra thirty-two bits of each floating point register.

Each individual register in the register set RFB[] may hold either a floating point value or an integer value. The register set RFB[] may include optional hardware for preventing accidental access of a floating point value as though it were an integer value, and vice versa. In one embodiment, however, in the interest of simplifying the register set RFB[], it is simply left to the software designer to ensure that no erroneous usages of individual registers are made. Thus, the execution engine 14 simply makes an access request on line 52, specifying an offset into the register set RFB[], without specifying whether the register at the given offset is intended to be used as a floating point register or an integer register. Within the execution engine 14, various entities may use either the full sixty-four bits provided by the register set RFB[], or may use only the low order thirty-two bits, such as in integer operations or single-precision floating point operations.

A first register RFB[0] 51 contains the constant value zero, in a form such that RB[0] is a thirty-two-bit integer zero (0000_{hex}) and RF[0] is a sixty-four-bit floating point zero (00000000_{hex}). This provides the same advantages as described above for RA[0].

C. Register Set C

The register set C 22 includes a plurality of Boolean registers 54 (RC[31:0]). RC[] is also known as the "condition status register" (CSR). The Boolean registers RC[] are each identical in structure and addressing, albeit that each is individually addressable at a unique address or offset within RC[].

In one embodiment, register set C further includes a "previous condition status register" (PCSR) 60, and the register set C also includes a CSR selector unit 62, which is responsive to the mode control unit 44 to select alternatively between the CSR 54 and the PCSR 60. In the one embodiment, the CSR is used when interrupts are enabled, and the PCSR is used when interrupts are disabled. The CSR and PCSR are identical in all other respects. In the one embodiment, when interrupts are set to be disabled, the CSR selector unit 62 pushes the contents of the CSR into the PCSR, overwriting the former contents of the PCSR, and when interrupts are re-enabled, the CSR selector unit 62 pops the contents of the PCSR back into the CSR. In other embodiments it may be desirable to merely alternate access between the CSR and the PCSR, as is done with RA[31:24] and RT[31:24]. In any event, the PCSR is always available as a thirty-two-bit "special register".

None of the Boolean registers is a dedicated condition flag, unlike the Boolean registers in previously known microprocessors. That is, the CSR 54 does not include a dedicated carry flag, nor a dedicated a minus flag, nor a dedicated flag indicating equality of a comparison or a zero subtraction result. Rather, any Boolean register may be the destination of the Boolean result of any Boolean operation. As with the other register sets, a first Boolean register 58 (RC[0]) always contains the value zero, to obtain the advantages explained above for RA[0]. In the preferred embodiment, each Boolean register is one bit wide, indicating one Boolean value.

II. EXECUTION ENGINE

The execution engine 14 includes one or more integer functional units 66, one or more floating point functional units 68, and one or more Boolean functional units 70. The functional units execute instructions as will be explained below. Buses 72, 73, and 75 connect the various elements of the IEU 10, and will each be understood to represent data, address, and control paths.

A. Instruction Format

Fig. 7 illustrates one exemplary format for an integer instruction which the execution engine 14 may execute. It will be understood that not all instructions need to adhere strictly to the illustrated format, and that the data processing system

includes an instruction fetcher and decoder (not shown) which are adapted to operate upon varying format instructions. The single example of Fig. 7 is for ease in explanation only. Throughout this Application the identification I[] will be used to identify various bits of the instruction. I[31:30] are reserved for future implementations of the execution engine 14. I[29:26] identify the instruction class of the particular instruction. Table 1 shows the various classes of instructions performed by the present invention.

TABLE 1
Instruction Classes

Class	Instructions
0-3	Integer and floating point register-to-register instructions
4	Immediate constant load
5	Reserved
6	Load
7	Store
8-11	Control Flow
12	Modifier
13	Boolean operations
14	Reserved
15	Atomic (extended)

Instruction classes of particular interest to this Application include the Class 0-3 register-to-register instructions and the Class 13 Boolean operations. While other classes of instructions also operate upon the register file 12, further discussion of those classes is not believed necessary in order to fully understand the present invention.

I[25] is identified as B0, and indicates whether the destination register is in register set A or register set B. I[24:22] are an opcode which identifies, within the given

instruction class, which specific function is to be performed. For example, within the register-to-register classes an opcode may specify "addition". I[21] identifies the addressing mode which is to be used when performing the instruction -- either register source addressing or immediate source addressing. I[20:16] identify the destination register as an offset within the register set indicated by B0. I[15] is identified as B1 and indicates whether the first operand is to be taken from register set A or register set B. I[14:10] identify the register offset from which the first operand is to be taken. I[9:8] identify a function selection -- an extension of the opcode I[24:22]. I[7:6] are reserved. I[5] is identified as B2 and indicates whether a second operand of the instruction is to be taken from register set A or register set B. Finally, I[4:0] identify the register offset from which the second operand is to be taken.

With reference to Fig. 1, the integer functional unit 66 and floating point functional unit 68 are equipped to perform integer comparison instructions and floating point comparisons, respectively. The instruction format for the comparison instruction is substantially identical to that shown in Fig. 7, with the caveat that various fields may advantageously be identified by slightly different names. I[20:16] identifies the destination register where the result is to be stored, but the addressing mode field I[21] does not select between register sets A or B. Rather, the addressing mode field indicates whether the second source of the comparison is found in a register or is immediate data. Because the comparison is a Boolean type

instruction, the destination register is always found in register set C. All other fields function as shown in Fig. 7. In performing Boolean operations within the integer and floating point functional units, the opcode and function select fields identify which Boolean condition is to be tested for in comparing the two operands. The integer and the floating point functional units fully support the IEEE standards for numerical comparisons.

The IEU 10 is a load/store machine. This means that when the contents of a register are stored to memory or read from memory, an address calculation must be performed in order to determine which location in memory is to be the source or the destination of the store or load, respectively. When this is the case, the destination register field I[20:16] identifies the register which is the destination or the source of the load or store, respectively. The source register 1 field, I[14:10], identifies a register in either set A or B which contains a base address of the memory location. In one embodiment, the source register 2 field, I[4:0], identifies a register in set A or set B which contains an index or an offset from the base. The load/store address is calculated by adding the index to the base. In another mode, I[7:0] include immediate data which are to be added as an index to the base.

B. <u>Operation of the Instruction Execution Unit and Register Sets</u>

It will be understood by those skilled in the art that the integer functional unit 66, the floating point functional unit

68, and the Boolean functional unit 70 are responsive to the contents of the instruction class field, the opcode field, and the function select field of a present instruction being executed.

1. <u>Integer Operations</u>

For example, when the instruction class, the opcode, and function select indicate that an integer register-to-register addition is to be performed, the integer functional unit may be responsive thereto to perform the indicated operation, while the floating point functional unit and the Boolean functional unit may be responsive thereto to not perform the operation. As will be understood from the cross-referenced applications, however, the floating point functional unit 68 is equipped to perform both floating point and integer operations. Also, the functional units are constructed to each perform more than one instruction simultaneously.

The integer functional unit 66 performs integer functions only. Integer operations typically involve a first source, a second source, and a destination. A given integer instruction will specify a particular operation to be performed on one or more source operands and will specify that the result of the integer operation is to be stored at a given destination. In some instructions, such as address calculations employed in load/store operations, the sources are utilized as a base and an index. The integer functional unit 66 is coupled to a first bus 72 over which the integer functional unit 66 is connected to

a switching and multiplexing control (SMC) unit A 74 and an SMC unit B 76. Each integer instruction executed by the integer functional unit 66 will specify whether each of its sources and destination reside in register set A or register set B.

Suppose that the IEU 10 has received, from the instruction fetch unit (not shown), an instruction to perform an integer register-to-register addition. In various embodiments, the instruction may specify a register bank, perhaps even a separate bank for each source and destination. In one embodiment, the instruction I[] is limited to a thirty-two-bit length, and does not contain any indication of which register bank 16-0 through 16-n is involved in the instruction. Rather, the bank selector unit 78 controls which register bank is presently active. In one embodiment, the bank selector unit 78 is responsive to one or more bank selection bits in a status word (not shown) within the IEU 10.

In order to perform the integer addition instruction, the integer functional unit 66 is responsive to the identification in I[14:10] and I[4:0] of the first and second source registers. The integer functional unit 66 places an identification of the first and second source registers at ports S1 and S2, respectively, onto the integer functional unit bus 72 which is coupled to both SMC units A and B 74 and 76. In one embodiment, the SMC units A and B are each coupled to receive B0-2 from the instruction I[]. In one embodiment, a zero in any respective Bn indicates register set A, and a one indicates register set B. During load/store operations, the source ports of the integer

and floating point functional units 66 and 68 are utilized as a base port and an index port, B and I, respectively.

After obtaining the first and second operands from the indicated register sets on the bus 72, as explained below, the integer functional unit 66 performs the indicated operation upon those operands, and provides the result at port D onto the integer functional unit bus 72. The SMC units A and B are responsive to B0 to route the result to the appropriate register set A or B.

The SMC unit B is further responsive to the instruction class, opcode, and function selection to control whether operands are read from (or results are stored to) either a floating point register RF[] or an integer register RB[]. As indicated, in one embodiment, the registers RF[] may be sixty-four bits wide while the registers are RB[] are only thirty-two bits wide. Thus, SMC unit B controls whether a word or a double word is written to the register set RFB[]. Because all registers within register set A are thirty-two bits wide, SMC unit A need not include means for controlling the width of data transfer on the bus 42.

All data on the bus 42 are thirty-two bits wide, but other sorts of complexities exist within register set A. The IEU mode integer switch 34 is responsive to the mode control unit 44 of the execution engine 14 to control whether data on the bus 42 are connected through to bus 36, bus 38 or bus 40, and vice versa.

IEU mode integer switch 34 is further responsive to I[20:16], I[14:10], and I[4:0]. If a given indicated destination or source is in RA[23:0], the IEU mode integer switch 34

automatically couples the data between lines 42 and 36. However, for registers RA[31:24], the IEU mode integer switch 34 determines whether data on line 42 is connected to line 38 or line 40, and vice versa. When interrupts are enabled, IEU mode integer switch 34 connects the SMC unit A to the second subset 28 of integer registers RA[31:24]. When interrupts are disabled, the IEU mode integer switch 34 connects the SMC unit A to the shadow registers RT[31:24]. Thus, an instruction executing within the integer functional unit 66 need not be concerned with whether to address RA[31:24] or RT[31:24]. It will be understood that SMC unit A may advantageously operate identically whether it is being accessed by the integer functional unit 66 or by the floating point functional unit 68.

2. Floating Point Operations

The floating point functional unit 68 is responsive to the class, opcode, and function select fields of the instruction, to perform floating point operations. The S1, S2, and D ports operate as described for the integer functional unit 66. SMC unit B is responsive to retrieve floating point operands from, and to write numerical floating point results to, the floating point registers RF[] on bus 52.

3. Boolean Operations

SMC unit C 80 is responsive to the instruction class, opcode, and function select fields of the instruction I[]. When SMC unit C detects that a comparison operation has been performed

by one of the numerical functional units 66 or 68, it writes the Boolean result over bus 56 to the Boolean register indicated at the D port of the functional unit which performed the comparison.

The Boolean functional unit 70 does not perform comparison instructions as do the integer and floating point functional units 66 and 68. Rather, the Boolean functional unit 70 is only used in performing bitwise logical combination of Boolean register contents, according to the Boolean functions listed in Table 2.

TABLE 2
Boolean Functions

I[23,22,9,8]	Boolean result calculation
0000	ZERO
0001	S1 AND S2
0010	S1 AND (NOT S2)
0011	S1
0100	(NOT S1) AND S2
0101	S2
0110	S1 XOR S2
0111	S1 OR S2
1000	S1 NOR S2
1001	S1 XNOR S2
1010	NOT S2
1011	S1 OR (NOT S2)
1100	NOT S1
1101	(NOT S1) OR S2
1110	S1 NAND S2
1111	ONE

The advantage which the present invention obtains by having a plurality of homogenous Boolean registers, each of which is individually addressable as the destination of a Boolean operation, will be explained with reference to Tables 3-5. Table 3 illustrates an example of a segment of code which performs a conditional branch based upon a complex Boolean function. The

complex Boolean function includes three portions which are OR-ed together. The first portion includes two sub-portions, which are AND-ed together.

TABLE 3
Example of Complex Boolean Function

```
1   RA[1] := 0;
2   IF (((RA[2] = RA[3]) AND (RA[4] > RA[5])) OR
3       (RA[6] < RA[7]) OR
4       (RA[8] <> RA[9])) THEN
5       X()
6   ELSE
7       Y();
8   RA[10] := 1;
```

Table 4 illustrates, in pseudo-assembly form, one likely method by which previous microprocessors would perform the function of Table 3. The code in Table 4 is written as though it were constructed by a compiler of at least normal intelligence operating upon the code of Table 3. That is, the compiler will recognize that the condition expressed in lines 2-4 of Table 3 is passed if any of the three portions is true.

TABLE 4
Execution of Complex Boolean Function
Without Boolean Register Set

```
1   START      LDI   RA[1],0
2   TEST1      CMP   RA[2],RA[3]
3              BNE   TEST2
4              CMP   RA[4],RA[5]
5              BGT   DO_IF
6   TEST2      CMP   RA[6],RA[7]
7              BLT   DO_IF
8   TEST3      CMP   RA[8],RA[9]
9              BEQ   DO_ELSE
10  DO_IF      JSR   ADDRESS OF X()
11             JMP   PAST_ELSE
12  DO_ELSE    JSR   ADDRESS OF Y()
13  PAST_ELSE  LDI   RA[10],1
```

The assignment at line 1 of Table 3 is performed by the "load immediate" statement at line 1 of Table 4. The first portion of the complex Boolean condition, expressed at line 2 of Table 3, is represented by the statements in lines 2-5 of Table 4. To test whether RA[2] equals RA[3], the compare statement at line 2 of Table 4 performs a subtraction of RA[2] from RA[3] or vice versa, depending upon the implementation, and may or may not store the result of that subtraction. The important function performed by the comparison statement is that the zero, minus, and carry flags will be appropriately set or cleared.

The conditional branch statement at line 3 of Table 4 branches to a subsequent portion of code upon the condition that RA[2] did not equal RA[3]. If the two were unequal, the zero flag will be clear, and there is no need to perform the second sub-portion. The existence of the conditional branch statement at line 3 of Table 4 prevents the further fetching, decoding, and

executing of any subsequent statement in Table 4 until the results of the comparison in line 2 are known, causing a pipeline stall. If the first sub-portion of the first portion (TEST1) is passed, the second sub-portion at line 4 of Table 4 then compares RA[4] to RA[5], again setting and clearing the appropriate status flags.

If RA[2] equals RA[3], and RA[4] is greater than RA[5], there is no need to test the remaining two portions (TEST2 and TEST3) in the complex Boolean function, and the statement at Table 4, line 5, will conditionally branch to the label DO_IF, to perform the operation inside the "IF" of Table 3. However, if the first portion of the test is failed, additional processing is required to determine which of the "IF" and "ELSE" portions should be executed.

The second portion of the Boolean function is the comparison of RA[6] to RA[7], at line 6 of Table 4, which again sets and clears the appropriate status flags. If the condition "less than" is indicated by the status flags, the complex Boolean function is passed, and execution may immediately branch to the DO_IF label. In various prior microprocessors, the "less than" condition may be tested by examining the minus flag. If RA[7] was not less than RA[6], the third portion of the test must be performed. The statement at line 8 of Table 4 compares RA[8] to RA[9]. If this comparison is failed, the "ELSE" code should be executed; otherwise, execution may simply fall through to the "IF" code at line 10 of Table 4, which is followed by an additional jump around the "ELSE" code. Each of the conditional

branches in Table 4, at lines 3, 5, 7 and 9, results in a separate pipeline stall, significantly increasing the processing time required for handling this complex Boolean function.

The greatly improved throughput which results from employing the Boolean register set C of the present invention will now readily be seen with specific reference to Table 5.

TABLE 5
Execution of Complex Boolean Function
With Boolean Register Set

1	START	LDI	RA[1],0
2	TEST1	CMP	RC[11],RA[2],RA[3],EQ
3		CMP	RC[12],RA[4],RA[5],GT
4	TEST2	CMP	RC[13],RA[6],RA[7],LT
5	TEST3	CMP	RC[14],RA[8],RA[9],NE
6	COMPLEX	AND	RC[15],RC[11],RC[12]
7		OR	RC[16],RC[13],RC[14]
8		OR	RC[17],RC[15],RC[16]
9		BC	RC[17],DO_ELSE
10	DO_IF	JSR	ADDRESS OF X()
11		JMP	PAST_ELSE
12	DO_ELSE	JSR	ADDRESS OF Y()
13	PAST_ELSE	LDI	RA[10],1

Most notably seen at lines 2-5 of Table 5, the Boolean register set C allows the microprocessor to perform the three test portions back-to-back without intervening branching. Each Boolean comparison specifies two operands, a destination, and a Boolean condition for which to test. For example, the comparison at line 2 of Table 5 compares the contents of RA[2] to the contents of RA[3], tests them for equality, and stores into RC[11] the Boolean value of the result of the comparison. Note that each comparison of the Boolean function stores its respective intermediate results in a separate Boolean register. As will be understood with reference to the above-referenced

related applications, the IEU 10 is capable of simultaneously performing more than one of the comparisons.

After at least the first two comparisons at lines 2-3 of Table 5 have been completed, the two respective comparison results are AND-ed together as shown at line 6 of Table 3. RC[15] then holds the result of the first portion of the test. The results of the second and third sub-portions of the Boolean function are OR-ed together as seen in Table 5, line 7. It will be understood that, because there are no data dependencies involved, the AND at line 6 and the OR-ed in line 7 may be performed in parallel. Finally, the results of those two operations are OR-ed together as seen at line 8 of Table 5. It will be understood that register RC[17] will then contain a Boolean value indicating the truth or falsity of the entire complex Boolean function of Table 3. It is then possible to perform a single conditional branch, shown at line 9 of Table 5. In the mode shown in Table 5, the method branches to the "ELSE" code if Boolean register RC[17] is clear, indicating that the complex function was failed. The remainder of the code may be the same as it was without the Boolean register set as seen in Table 4.

The Boolean functional unit 70 is responsive to the instruction class, opcode, and function select fields as are the other functional units. Thus, it will be understood with reference to Table 5 again, that the integer and/or floating point functional units will perform the instructions in lines 1-5 and 13, and the Boolean functional unit 70 will perform the

Boolean bitwise combination instructions in lines 6-8. The control flow and branching instructions in line 9-12 will be performed by elements of the IEU 10 which are not shown in Fig. 1.

III. DATA PATHS

Figs. 2-5 illustrate further details of the data paths within the floating point, integer, and Boolean portions of the IEU, respectively.

A. Floating Point Portion Data Paths

As seen in Fig. 2, the register set FB 20 is a multi-ported register set. In one embodiment, the register set FB 20 has two write ports WFB0-1, and five read ports RDFB0-4. The floating point functional unit 68 of Fig. 1 is comprised of the ALU2 102, FALU 104, MULT 106, and NULL 108 of Fig. 2. All elements of Fig. 2 except the register set 20 and the elements 102-108 comprise the SMC unit B of Fig. 1.

External, bidirectional data bus EX_DATA[] provides data to the floating point load/store unit 122. Immediate floating point data bus LDF_IMED[] provides data from a "load immediate" instruction. Other immediate floating point data are provided on busses RFF1_IMED and RFF2_IMED, such as is involved in an "add immediate" instruction. Data are also provided on bus EX_SR_DT[], in response to a "special register move" instruction. Data may also arrive from the integer portion, shown in Fig. 3, on busses 114 and 120.

The floating point register set's two write ports WFB0 and WFB1 are coupled to write multiplexers 110-0 and 110-1, respectively. The write multiplexers 110 receive data from: the ALU0 or SHF0 of the integer portion of Fig. 3; the FALU; the MULT; the ALU2; either EX_SR_DT[] or LDF_IMED[]; and EX_DATA[]. Those skilled in the art will understand that control signals (not shown) determine which input is selected at each port, and address signals (not shown) determine to which register the input data are written. Multiplexer control and register addressing are within the skill of persons in the art, and will not be discussed for any multiplexer or register set in the present invention.

The floating point register set's five read ports RDFB0 to RDFB4 are coupled to read multiplexers 112-0 to 112-4, respectively. The read multiplexers each also receives data from: either EX_SR_DT[] or LDF_IMED[], on load immediate bypass bus 126; a load external data bypass bus 127, which allows external load data to skip the register set FB; the output of the ALU2 102, which performs non-multiplication integer operations; the FALU 104, which performs non-multiplication floating point operations; the MULT 106, which performs multiplication operations; and either the ALU0 140 or the SHF0 144 of the integer portion shown in Fig. 3, which respectively perform non-multiplication integer operations and shift operations. Read multiplexers 112-1 and 112-3 also receive data from RFF1_IMED[] and RFF2_IMED[], respectively.

Each arithmetic-type unit 102-106 in the floating point portion receives two inputs, from respective sets of first and second source multiplexers S1 and S2. The first source of each unit ALU2, FALU, and MULT comes from the output of either read multiplexer 112-0 or 112-2, and the second source comes from the output of either read multiplexer 112-1 or 112-3. The sources of the FALU and the MULT may also come from the integer portion of Fig. 3 on bus 114.

The results of the ALU2, FALU, and MULT are provided back to the write multiplexers 110 for storage into the floating point registers RF[], and also to the read multiplexers 112 for re-use as operands of subsequent operations. The FALU also outputs a signal FALU_BD indicating the Boolean result of a floating point comparison operation. FALU_BD is calculated directly from internal zero and sign flags of the FALU.

Null byte tester NULL 108 performs null byte testing operations upon an operand from a first source multiplexer, in one mode that of the ALU2. NULL 108 outputs a Boolean signal NULLB_BD indicating whether the thirty-two-bit first source operand includes a byte of value zero.

The outputs of read multiplexers 112-0, 112-1, and 112-4 are provided to the integer portion (of Fig. 3) on bus 118. The output of read multiplexer 112-4 is also provided as STDT_FP[] store data to the floating point load/store unit 122.

Fig. 5 illustrates further details of the control of the S1 and S2 multiplexers. As seen, in one embodiment, each S1 multiplexer may be responsive to bit B1 of the instruction I[],

and each S2 multiplexer may be responsive to bit B2 of the instruction I[]. The S1 and S2 multiplexers select the sources for the various functional units. The sources may come from either of the register files, as controlled by the B1 and B2 bits of the instruction itself. Additionally, each register file includes two read ports from which the sources may come, as controlled by hardware not shown in the Figs.

B. Integer Portion Data Paths

As seen in Fig. 3, the register set A 18 is also multi-ported. In one embodiment, the register set A 18 has two write ports WA0-1, and five read ports RDA0-4. The integer functional unit 66 of Fig. 1 is comprised of the ALU0 140, ALU1 142, SHF0 144, and NULL 146 of Fig. 3. All elements of Fig. 3 except the register set 18 and the elements 140-146 comprise the SMC unit A of Fig. 1.

External data bus EX_DATA[] provides data to the integer load/store unit 152. Immediate integer data on bus LDI_IMED[] are provided in response to a "load immediate" instruction. Other immediate integer data are provided on busses RFA1_IMED and RFA2_IMED in response to non-load immediate instructions, such as an "add immediate". Data are also provided on bus EX_SR_DT[] in response to a "special register move" instruction. Data may also arrive from the floating point portion (shown in Fig. 2) on busses 116 and 118.

The integer register set's two write ports WA0 and WA1 are coupled to write multiplexers 148-0 and 148-1, respectively. The write multiplexers 148 receive data from: the FALU or MULT of the floating point portion (of Fig. 2); the ALU0; the ALU1; the SHF0; either EX_SR_DT[] or LDI_IMED[]; and EX_DATA[].

The integer register set's five read ports RDA0 to RDA4 are coupled to read multiplexers 150-0 to 150-4, respectively. Each read multiplexer also receives data from: either EX_SR_DT[] or LDI_IMED[] on load immediate bypass bus 160; a load external data bypass bus 154, which allows external load data to skip the register set A; ALU0; ALU1; SHF0; and either the FALU or the MULT of the floating point portion (of Fig. 2). Read multiplexers 150-1 and 150-3 also receive data from RFA1_IMED[] and RFA2_IMED[], respectively.

Each arithmetic-type unit 140-144 in the integer portion receives two inputs, from respective sets of first and second source multiplexers S1 and S2. The first source of ALU0 comes from either the output of read multiplexer 150-2, or a thirty-two-bit wide constant zero (0000_{hex}), or floating point read multiplexer 112-4. The second source of ALU0 comes from either read multiplexer 150-3 or floating point read multiplexer 112-1. The first source of ALU1 comes from either read multiplexer 150-0 or IF_PC[]. IF_PC[] is used in calculating a return address needed by the instruction fetch unit (not shown), due to the IEU's ability to perform instructions in an out-of-order sequence. The second source of ALU1 comes from either read multiplexer 150-1 or CP_OFFSET[]. CP_OFFSET[] is used in calculating a return address for a CALL instruction, also due to the out-of-order capability.

The first source of the shifter SHF0 144 is from either: floating point read multiplexer 112-0 or 112-4; or any integer read multiplexer 150. The second source of SHF0 is from either: floating point read multiplexer 112-0 or 112-4; or integer read multiplexer 150-0, 150-2, or 150-4. SHF0 takes a third input from a shift amount multiplexer (SA). The third input controls how far to shift, and is taken by the SA multiplexer from either: floating point read multiplexer 112-1; integer read multiplexer 150-1 or 150-3; or a five-bit wide constant thirty-one (11111_2 or 31_{10}). The shifter SHF0 requires a fourth input from the size multiplexer (S). The fourth input controls how much data to shift, and is taken by the S multiplexer from either: read multiplexer 150-1; read multiplexer 150-3; or a five-bit wide constant sixteen (10000_2 or 16_{10}).

The results of the ALU0, ALU1, and SHF0 are provided back to the write multiplexers 148 for storage into the integer registers RA[], and also to the read multiplexers 150 for re-use as operands of subsequent operations. The output of either ALU0 or SHF0 is provided on bus 120 to the floating point portion of Fig. 3. The ALU0 and ALU1 also output signals ALU0_BD and ALU1_BD, respectively, indicating the Boolean results of integer comparison operations. ALU0_BD and ALU1_BD are calculated directly from the zero and sign flags of the respective functional units. ALU0 also outputs signals EX_TADR[] and EX_VM_ADR. EX_TADR[] is the target address generated for an absolute branch instruction, and is sent to the IFU (not shown) for fetching the target instruction. EX_VM_ADR[] is the virtual

address used for all loads from memory and stores to memory, and is sent to the VMU (not shown) for address translation.

Null byte tester NULL 146 performs null byte testing operations upon an operand from a first source multiplexer. In one embodiment, the operand is from the ALU0. NULL 146 outputs a Boolean signal NULLA_BD indicating whether the thirty-two-bit first source operand includes a byte of value zero.

The outputs of read multiplexers 150-0 and 150-1 are provided to the floating point portion (of Fig. 2) on bus 114. The output of read multiplexer 150-4 is also provided as STDT_INT[] store data to the integer load/store unit 152.

A control bit PSR[7] is provided to the register set A 18. It is this signal which, in Fig. 1, is provided from the mode control unit 44 to the IEU mode integer switch 34 on line 46. The IEU mode integer switch is internal to the register set A 18 as shown in Fig. 3.

Fig. 6 illustrates further details of the control of the S1 and S2 multiplexers. The signal ALU0_BD

C. <u>Boolean Portion Data Paths</u>

As seen in Fig. 4, the register set C 22 is also multi-ported. In one embodiment, the register set C 22 has two write ports WC0-1, and five read ports RDA0-4. All elements of Fig. 4 except the register set 22 and the Boolean combinational unit 70 comprise the SMC unit C of Fig. 1.

The Boolean register set's two write ports WC0 and WC1 are coupled to write multiplexers 170-0 and 170-1, respectively. The

write multiplexers 170 receive data from: the output of the Boolean combinational unit 70, indicating the Boolean result of a Boolean combinational operation; ALU0_BD from the integer portion of Fig. 3, indicating the Boolean result of an integer comparison; PALU_BD from the floating point portion of Fig. 2, indicating the Boolean result of a floating point comparison; either ALU1_BD_P from ALU1, indicating the results of a compare instruction in ALU1, or NULLA_BD from NULL 146, indicating a null byte in the integer portion; and either ALU2_BD_P from ALU2, indicating the results of a compare operation in ALU2, or NULLB_BD from NULL 108, indicating a null byte in the floating point portion. In one mode, the ALU0_BD, ALU1_BD, ALU2_BD, and PALU_BD signals are not taken from the data paths, but are calculated as a function of the zero flag, minus flag, carry flag, and other condition flags in the PSR. In one mode, wherein up to eight instructions may be executing at one instant in the IEU, the IEU maintains up to eight PSRs.

The Boolean register set C is also coupled to bus EX_SR_DT[], for use with "special register move" instructions. The CSR may be written or read as a whole, as though it were a single thirty-two-bit register. This enables rapid saving and restoration of machine state information, such as may be necessary upon certain drastic system errors or upon certain forms of grand scale context switching.

The Boolean register set's five read ports RDC0 to RDC3 are coupled to read multiplexers 172-0 to 172-4, respectively. The read multiplexers 172 receive the same set of inputs as the write

multiplexers 170 receive. The Boolean combinational unit 70 receives inputs from read multiplexers 170-0 and 170-1. Read multiplexers 172-2 and 172-3 respectively provide signals BLBP_CPORT and BLBP_DPORT. BLBP_CPORT is used as the basis for conditional branching instructions in the IEU. BLBP_DPORT is used in the "add with Boolean" instruction, which sets an integer register in the A or B set to zero or one (with leading zeroes), depending upon the content of a register in the C set. Read port RDC4 is presently unused, and is reserved for future enhancements of the Boolean functionality of the IEU.

IV. CONCLUSION

While the features and advantages of the present invention have been described with respect to particular embodiments thereof, and in varying degrees of detail, it will be appreciated that the invention is not limited to the described embodiments. The following Claims define the invention to be afforded patent coverage.

CLAIMS

We claim:

1. An apparatus executing a set of instructions, the instructions including one or more fields, wherein a field of a given instruction specifies a source of an operand of the given instruction or a destination of a result of the given instruction, and wherein the apparatus comprises:

processing means for executing the instructions; and

a register file, coupled to the processing means, for storing operands and results of the instructions, wherein,

the register file includes a plurality of register sets, and

the register file is responsive to one or more of the fields in a given instruction to retrieve an operand of the given instruction from, or store a result of the given instruction into, a given register in a given one of the register sets as identified by the one or more fields in the given instruction.

2. The apparatus of Claim 1, wherein the instructions include Boolean combinational instructions each operating on one or more Boolean operands to generate a Boolean result, each Boolean combinational instruction including one or more Boolean fields specifying a location of each operand and result, and wherein:

the processing means includes Boolean execution means for executing the Boolean combinational instructions;

the register file includes a Boolean register set of Boolean registers, each Boolean register for holding one of said Boolean operands or Boolean results; and

the register file is responsive to each said Boolean field in a given Boolean combinational instruction independent of what Boolean combinational operation is specified by the given Boolean combinational instruction.

3. The apparatus of Claim 2, wherein the instructions include Boolean comparison instructions each operating on one or more operands to generate a Boolean result, each Boolean comparison instruction including a Boolean result field specifying a location, in the Boolean register set, of the Boolean result, and wherein:

the processing means includes comparison means for executing the Boolean comparison instructions; and

the register file is responsive to the Boolean result field in a given Boolean instruction independent of what Boolean comparison operation is specified by the given Boolean comparison instruction.

4. The apparatus of Claim 1, wherein the instructions include integer instructions each operating on one or more integer operands to generate an integer result, each integer

instruction including one or more integer fields specifying a location of each operand and result, and wherein:

the processing means includes integer execution means for executing the integer instructions; and

the register file includes an integer register set of integer registers, each integer register for holding one of said integer operands or integer results.

5. The apparatus of Claim 4, wherein the register file further comprises:

a plurality of integer register sets.

6. The apparatus of Claim 1, wherein the instructions include floating point instructions each operating on one or more floating point operands to generate a floating point result, each floating point instruction including one or more floating point fields specifying a location of each operand and result, and wherein:

the processing means includes floating point execution means for executing the floating point instructions; and

the register file includes a floating point register set of floating point registers, each floating point register for holding one of said floating point operands or floating point results.

7. An apparatus comprising:

means for executing Boolean instructions, the Boolean instructions performing Boolean operations upon operands to generate Boolean results and each Boolean instruction indicating a destination for storage of the Boolean results of the Boolean instruction;

a plurality of Boolean register means each for holding a Boolean value; and

means, responsive to execution of a given Boolean instruction by said means for executing, for storing the given Boolean instruction's Boolean result into one of said Boolean register means, the one Boolean register means being indicated by said given Boolean instruction as the destination of its Boolean result.

8. The apparatus of Claim 7, wherein the means for executing Boolean instructions comprises:

numerical execution means for executing numerical comparison instructions to compare two multi-bit numerical operands and to accordingly produce a single-bit Boolean value result.

9. The apparatus of Claim 8, wherein the numerical execution means comprises:

integer execution means for comparing two multi-bit integer operands.

10. The apparatus of Claim 8, wherein the numerical execution means comprises:

floating point execution means for comparing two multi-bit floating point operands.

11. The apparatus of Claim 10, wherein the numerical execution means further comprises:

integer execution means for comparing two multi-bit integer operands.

12. The apparatus of Claim 7, wherein the means for executing Boolean instructions comprises:

Boolean execution means for executing Boolean combinational instructions to combine two Boolean value operands and to accordingly produce a single-bit Boolean value result.

13. The apparatus of Claim 12, wherein the means for executing Boolean instructions further comprises:

numerical execution means for executing numerical comparison instructions to compare two multi-bit numerical operands and to accordingly produce a single-bit Boolean value result.

14. The apparatus of Claim 13, wherein the numerical execution means comprises:

integer execution means for comparing two multi-bit integer operands; and

floating point execution means for comparing two multi-bit floating point operands.

15. The apparatus of Claim 7 further comprising:

numerical register means for holding integer and floating point values;

numerical execution means for executing numerical comparison instructions, wherein execution of each given numerical comparison instruction,

i) retrieves two or more multi-bit numerical operands from respective numerical register means specified by the given numerical comparison instruction,

ii) compares the two or more numerical operands according to a condition specified by the given numerical comparison instruction,

iii) produces a first single-bit Boolean value result according to the condition,

iv) stores the first Boolean value result in a given one of said Boolean register means as specified by the given numerical comparison instruction,

wherein the numerical execution means includes,

i) integer execution means for comparing two multi-bit integer operands, and

ii) floating point execution means for comparing two multi-bit floating point operands; and

Boolean execution means for executing Boolean combinational instructions, wherein execution of each given Boolean combinational instruction,

i) retrieves one or more Boolean value operands from respective Boolean register means as specified by the given Boolean combinational instruction,

ii) combines the one or more Boolean value operands according to an operation specified by the given Boolean combinational instruction,

iii) produces a second single-bit Boolean value result according to the operation, and

iv) stores the second Boolean value result in a given one of said Boolean register means as specified by the given Boolean combinational instruction.

16. The apparatus of Claim 7, wherein:

the plurality of Boolean register means includes,

i) a first set of Boolean registers, and

ii) a second set of Boolean registers; and the apparatus further comprises

means, coupled to the plurality of Boolean register means, for selecting the first or the second set of Boolean registers as a currently active set; and

the means for storing is responsive to the means for selecting, to store results into Boolean registers in the currently active set only.

17. An apparatus for use with a data processing system, the data processing system including means for executing Boolean instructions, each Boolean instruction performing a given Boolean operation upon two or more operands to generate a one-bit Boolean result, the apparatus comprising:

a Boolean register set including a plurality of individually addressable one-bit registers; and

control means for writing the one-bit result of a given Boolean instruction into one of said one-bit registers, the one one-bit register being specified by the given Boolean instruction's contents.

18. The apparatus of Claim 17, wherein the Boolean instructions include Boolean combinational instructions, each Boolean combinational instruction specifying a Boolean operation to be performed upon a first and a second operand to generate the result, and specifying a first address of the first operand and a second address of the second operand and a third address of a destination for the result, wherein:

the control means is further for reading the first and second operands from the Boolean register set at the first and second addresses, respectively, and wherein the one one-bit register is specified by the third address.

19. The apparatus of Claim 18, wherein the means for executing includes means for executing plural Boolean instructions in parallel, wherein there may exist, in the plural

Boolean instructions, data dependency between one or more slave instructions and a master instruction, each slave instruction having the result of the master instruction as an operand such that the slave instruction cannot be executed until the result of the master instruction has been generated, the means for executing further includes means for delaying data dependent instructions until their dependent data supplying instruction is completed and its result is generated, and wherein:

a prespecified constant Boolean register of the one-bit registers has a predetermined constant data value which does not change upon the control means writing another value to the prespecified constant Boolean register; and

the control means is responsive to a master instruction whose destination is the prespecified constant Boolean register, to immediately read the predetermined constant data value for supply to the slave instructions, whereby the means for executing is enabled to execute the slave instructions before the result of the master instruction is generated.

20. An apparatus comprising:

execution means for executing instructions, the instructions performing operations upon operands to generate results, each instruction specifying a respective source address for each operand and a destination address for the result of the instruction, each address specifying a register set and an offset;

a first register set including a plurality of individually addressable registers each for storing a value of a first data type;

first access means for writing and reading values to and from the first register set according to a given instruction, the first access means including,

i) first reading means, responsive to the given instruction having a given source address which specifies the first register set as a source for an operand of the given instruction, for reading the operand's value from the first register set at the offset specified by the given source address, and

ii) first writing means, responsive to the given instruction having a given destination address which specifies the first register set as a destination for the result of the given instruction, for writing the result's value to the first register set at the offset specified by the given destination address;

a second register set including a plurality of individually addressable registers each for storing a value of the first data type; and

second access means for writing and reading values to and from the second register set according to the given instruction, the second access means including,

i) second reading means, responsive to the given instruction having a given source address which specifies the second register set as a source for an operand of the given instruction, for reading the operand's value from the second register set at the offset specified by the given source address, and

ii) second writing means, responsive to the given instruction having a given destination address which specifies the second register set as a destination for the result of the given instruction, for writing the result's value to the second register set at the offset specified by the given destination address.

21. The apparatus of Claim 20, wherein:

a given instruction may specify a first and a second source address and a destination address, with each address specifying either of the first or second register sets such that the given instruction requires access to both register sets; and

the first and second access means operate simultaneously to provide the instruction parallel access to both the first and second register sets.

22. In a data processing system, which includes a central processing unit (CPU) which performs operations according to an instruction, the operations operating upon data of a first data type, a data register system comprising:

a first register set including a plurality of first registers each for holding a datum of the first data type, and including means for accessing the first registers in response to the instruction; and

a second register set including a plurality of second registers each for holding a datum of the first data type, and including means for accessing the second registers in response to the instruction.

23. The data register system of Claim 22, wherein the instruction includes a field specifying which of the first and second register sets is to be accessed in response to the instruction, and wherein the data register system further comprises:

means, responsive to the field, for accessing the first register set or the second register set as specified by the field.

24. An apparatus comprising:

integer execution means for executing integer instructions, each integer instruction performing an integer operation upon one or more integer value operands and generating an integer value result;

floating point execution means for executing floating point instructions, each floating point operation performing a floating point operation upon one or more floating point value operands and generating a floating point value result;

wherein each instruction specifies one or more sources from which its one or more operands are to be retrieved and further specifies a destination to which its result is to be stored, each operation also optionally specifying an integer value base and an integer value index;

a register bank including,

i) first register set means, having a plurality of first registers, for holding integer values and floating point values;

access means, coupled to the first register set means and to both execution means, for,

i) retrieving, from any one first register, an integer value operand for the integer execution means, a floating point value operand for the floating point execution means, or an integer value base or index for either execution means, as indicated by an instruction, and

ii) for storing, into any one first register, an integer value result from the integer execution means or a

floating point value result from the floating point execution means, as indicated by an instruction.

25. The apparatus of Claim 24, wherein:

the register bank further comprises second register set means, having a plurality of second registers, for holding integer values; and

the access means is further for,

i) retrieving, from any one second register, an integer value operand for the integer execution means, or an integer value base or index for either execution means, as indicated by an instruction, and

ii) for storing, into any one second register, an integer value result from the integer execution means, as indicated by an instruction.

26. The apparatus of Claim 25, further comprising:

Boolean execution means for executing Boolean combinational instructions, each Boolean combinational instruction performing a Boolean combinational operation upon one or more Boolean value operands and generating a Boolean value result;

the register bank further comprises third register set means, having a plurality of third registers, for holding Boolean values; and

the access means is further for,

i) retrieving, from any one third register, a Boolean value operand for the Boolean execution means, as indicated by a Boolean combinational instruction, and

ii) for storing, into any one third register, a Boolean value result from the Boolean execution means, as indicated by a Boolean combinational instruction.

27. An apparatus, for use with a data processing system which performs read operations and write operations upon data values of a first data type and a first data width and upon data values of a second data type and a second data width different than the first data width, the data processing system specifying a read address and data type for each read and a write address and data content for each write, the apparatus comprising:

a register set including a plurality of individually addressable registers, each register being wide enough to hold a value of either data width;

read access means, responsive to the data processing system performing a given read operation, for accessing the register set to retrieve data contents of a given register, which is individually addressed at the given read operation's specified read address, and for providing to the data processing system such portion of the retrieved data contents as the data type of the read operation specifies; and

write access means, responsive to the data processing system performing a given write operation, for accessing the register set to store into a given register, which is individually addressed at the given write operation's specified write address, the data content specified by the write operation.

28. The apparatus of Claim 27, wherein the first data type is floating point, the first data width is sixty-four bits, the second data type is integer, the second data width is thirty-two bits, and wherein:

the register set is sixty-four bits wide; and

the read and write access means respectively retrieve and store sixty-four bits responsive to the data processing system performing floating point operations, and thirty-two bits responsive to the data processing system performing integer operations.

29. An apparatus for use with a data processing system which executes instructions, each instruction performing operations upon one or more operands and generating a result, wherein each instruction specifies one or more sources from which its one or more operands are to be retrieved and further specifies a destination to which its result is to be stored, wherein the data processing system operates in a plurality of modes, the apparatus comprising:

a plurality of first register means each for holding an operand or a result;

a plurality of second register means each for holding an operand or a result; and

switch means, responsive to the mode of the data processing system, for providing the data processing system access to only the plurality of first register means when the data processing system operates in a first mode, and for providing the data processing system access to only a first subset of the plurality of first register means and to the plurality of second register means when the data processing system operates in a second mode.

30. An apparatus including execution means for executing instructions, each instruction performing operations on one or more operands and generating a result, each instruction specifying one or more sources which are to be accessed to read its one or more operands and a destination which is to be accessed to write its result, the apparatus further comprising:

a plurality of register banks;

each register bank including a plurality of register means, each register means for storing an operand or a result, the plurality of register means within each register bank being arranged in a sequence such that any one given register means within a given register bank may be accessed as an offset into the given register bank, wherein the sources and the destination of a given instruction are specified as offsets; and

register bank selector means for selecting a given register bank into which the given instruction's source and destination offsets are applied, the register bank selector means operating independently of any contents of the given instruction.

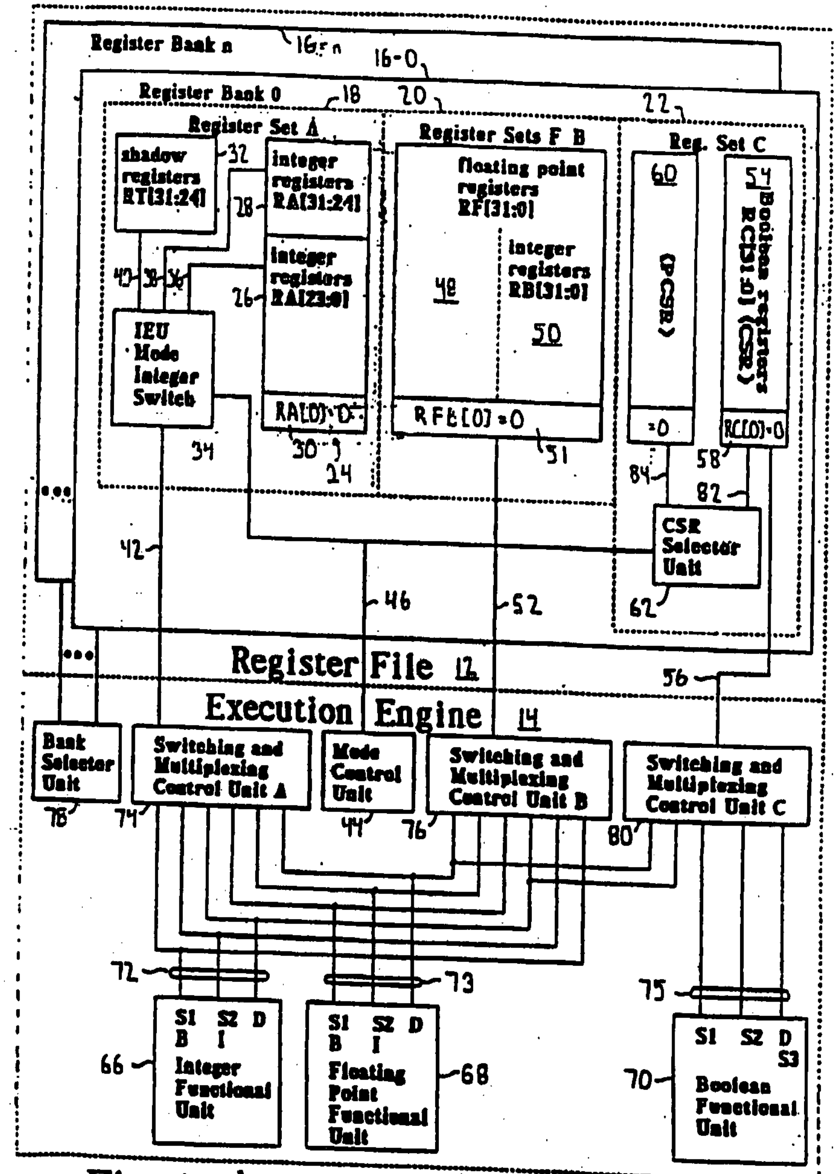


Fig. 1 10 Instruction Execution Unit (IEU)

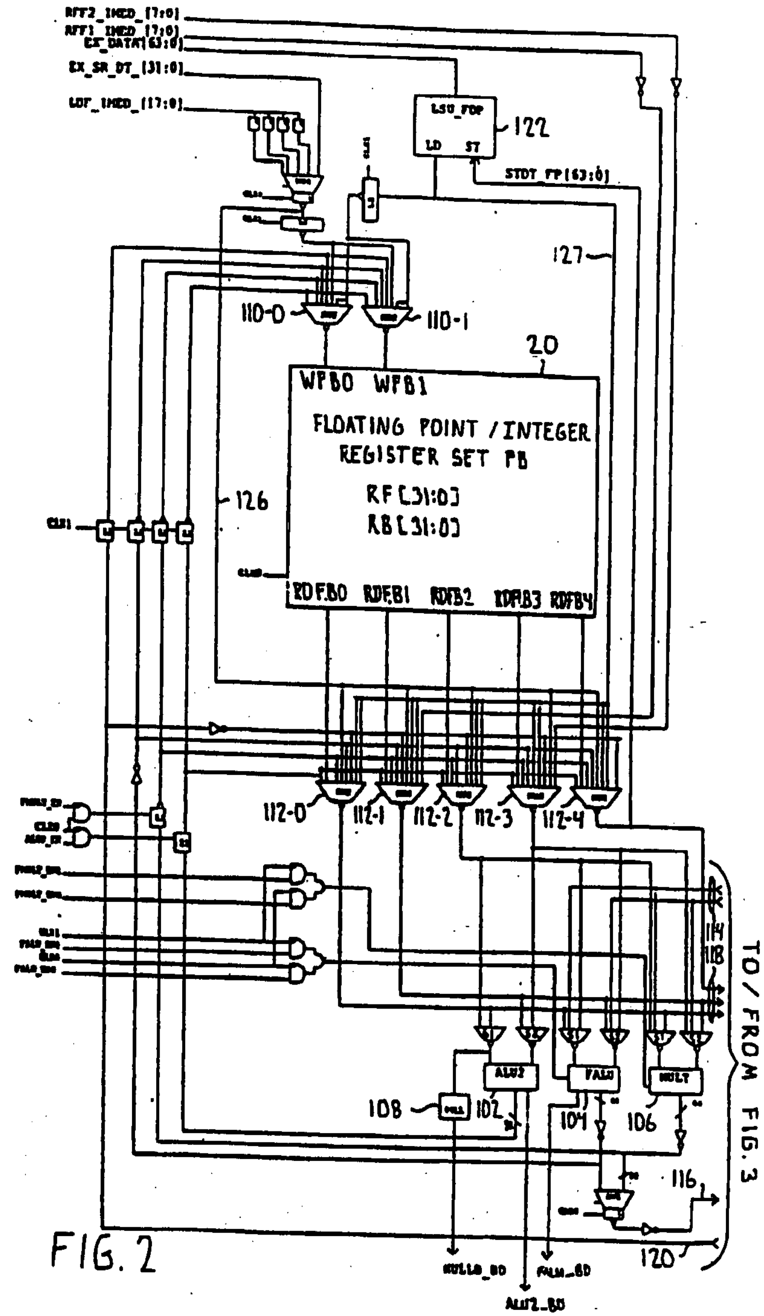

RFF2_IMED_[7:0]
RFF1_IMED_[7:0]
EX_DATA[63:0]
EX_SR_DT_[31:0]
LDT_IMED_[17:0]
LSU_FOP
LD
ST
122
STDT_FP[63:0]
127
110-0
110-1
20
WFB0 WFB1
FLOATING POINT / INTEGER
REGISTER SET FB
RF[31:0]
RB[31:0]
126
RDFB0 RDFB1 RDFB2 RDFB3 RDFB4
112-0
112-1
112-2
112-3
112-4
114
118
TO / FROM FIG. 3
ALU2
FALU
MULT
108
102
104
106
116
120
ALU2_BD
FALU_BD

FIG. 2

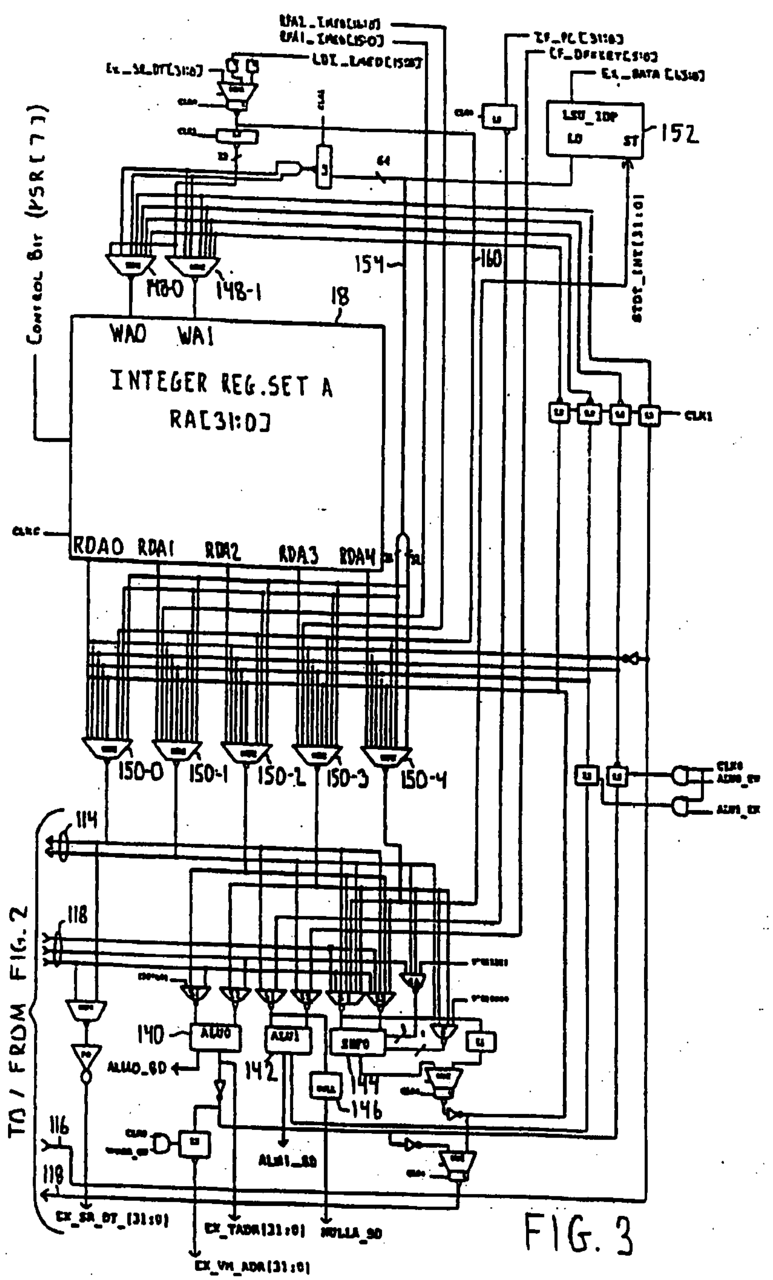

INTEGER REG. SET A
RA[31:0]
WA0
WA1
RDA0
RDA1
RDA2
RDA3
RDA4
18
148-0
148-1
154
160
152
LSU_IDP
LD
ST
64
150-0
150-1
150-2
150-3
150-4
114
118
116
119
140
ALU0
ALU1
142
SHF0
144
146
ALU0_SD
ALU1_SD
NULLA_SD
EX_SR_DT_[31:0]
EX_TADR[31:0]
EX_VM_ADR[31:0]
CLK1
Control Bit (PSR[7])
TO / FROM FIG. 2

FIG. 3

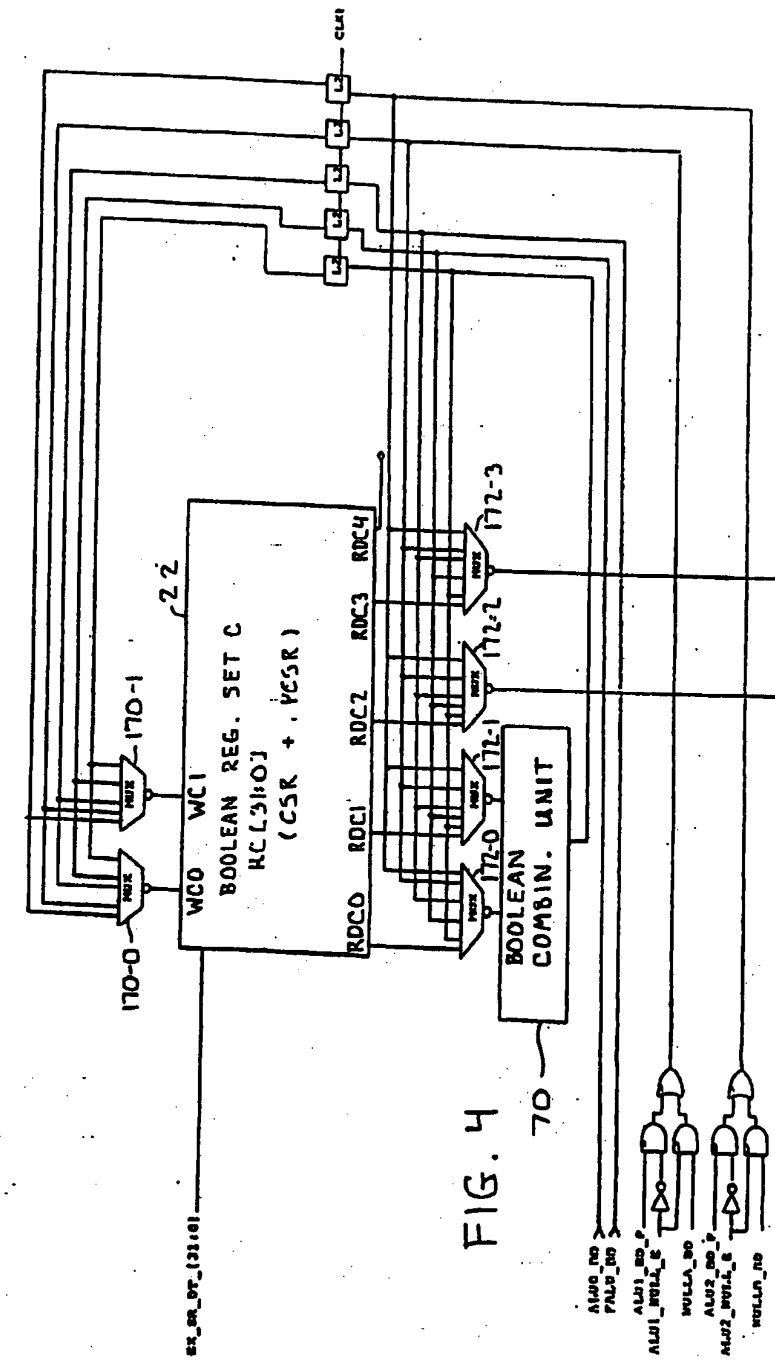

BOOLEAN REG. SET C
RC[31:0]
(CSR + PCSR)
WC0
WC1
RDC0
RDC1
RDC2
RDC3
RDC4
22
170-0
170-1
172-0
172-1
172-2
172-3
BOOLEAN COMBIN. UNIT
70

FIG. 4

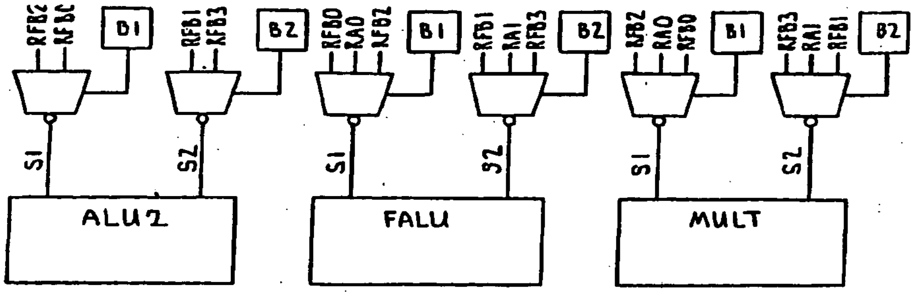


FIG. 5

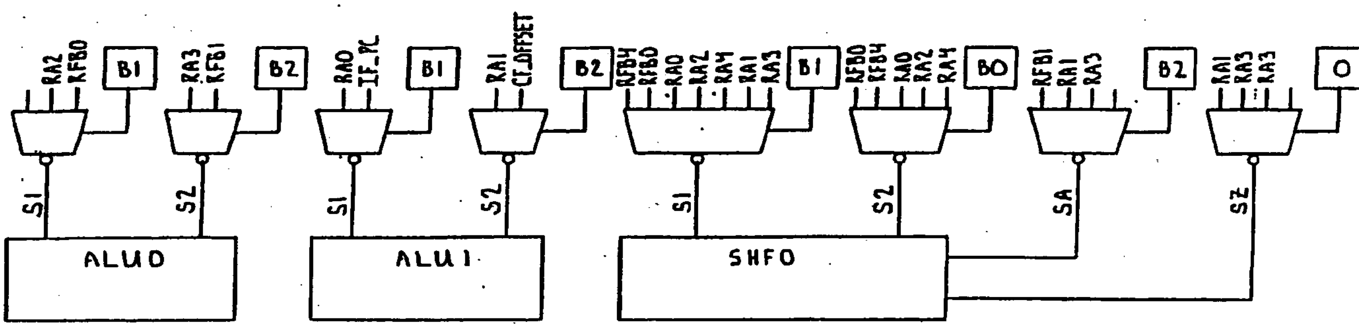


FIG. 6

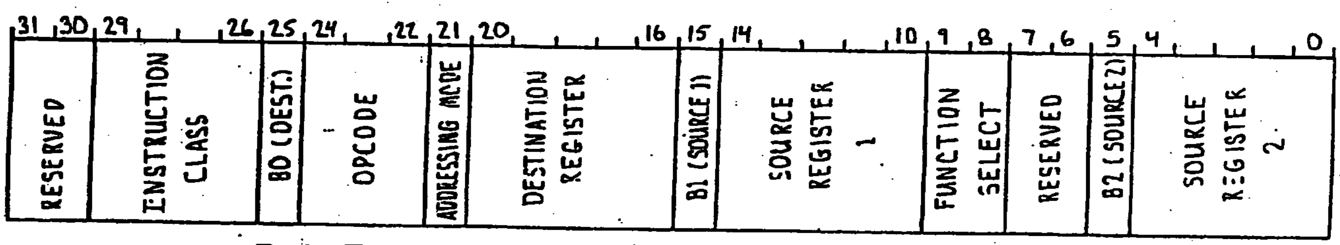


FIG. 7

INSTRUCTION I [ ]

INTERNATIONAL SEARCH REPORT

International Application No PCT/US 92/05720

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all)[6]

According to International Patent Classification (IPC) or to both National Classification and IPC

Int.Cl. 5 G06F9/30

II. FIELDS SEARCHED

Minimum Documentation Searched[7]

Classification System	Classification Symbols
Int.Cl. 5	G06F

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched[8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT[9]

Category°	Citation of Document,[11] with indication, where appropriate, of the relevant passages[12]	Relevant to Claim No.[13]
X Y A	GB,A,2 190 521 (INTEL CORPORATION) 18 November 1987 * page 2, lines 16-28; page 2, line 42 - page 3, line 44 *	1,4-6, 24,25,27 15,26 28
X	EP,A,0 213 843 (ADVANCED MICRO DEVICES INC.) 11 March 1987 * page 2, line 27 - page 3, line 6; page 5, lines 2-19; page 7, line 22 - page 8, line 12 *	1,22,29

-/--

° Special categories of cited documents :[10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

Date of the Actual Completion of the International Search	Date of Mailing of this International Search Report
16 OCTOBER 1992	2?.10.92
International Searching Authority	**Signature of Authorized Officer**
EUROPEAN PATENT OFFICE	WEINBERG L.F.

Form PCT/ISA/210 (second sheet) (January 1985)

International Application No PCT/US 92/05720

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

Category°	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No.
X	PROCEEDINGS FALL JOINT COMPUTER CONFERENCE NOVEMBER 2-6,1986,DALLAS,US; p1014-1019; MAEJIME ET AL:'A 16-bit microprocessor with multi-register bank architecture', * page 1015, left column, lines 15-31; page 1016, left column, line 1 - page 1017, left column, line 13 *	1,22,30
X	MICROPROCESSORS AND MICROSYSTEMS vol. 14, no. 6, July 1990, LONDON GB pages 333 - 340 MILLER AND QUAMMEN 'Exploiting large register sets'	1,20-23
A	* page 334. right column, line 54 - page 335, left column, line 30 *	16
X	EP,A,0 170 284 (NEC CORPORATION) 5 February 1986	1,20-23
Y	* abstract; column 2, line 1 - column 2, line 40; column 3, lines 47-60; column 5, line 28 - column 7, line 18 *	2,3
X	MICROPROCESSING AND MICROPROGRAMMING. vol. 29, no. 3, October 1990, AMSTERDAM NL pages 137 - 149 ADAMS ET AL. 'Utilising low level parallelism in general purpose code: the HARP project'	7-9,12, 13,17,18
Y	* page 138, right column, line 30-42; page 139, left column, line 3 - right column, line 3; page 140, left column, line 9 - right column, line 12 *	2,3,10, 11,14, 15,26
Y	35TH IEEE COMPUTER SOCIETY INTERNATIONAL CONFERENCE, COMPCON SPRING '90, 26 February 1990, SAN FRANCISCO,US pages 166 - 172 GROVES AND OEHLER 'An IBM second generation RISC processor architecture'	10,11,14
X	US,A,4 212 076 (CONNERS) 8 July 1980 * abstract; column 1, line 40 - column 3, line 14; column 6, line 5 - column 7, line 3; column 13, line 15 - column 16, line 54 *	7,12,17, 18
P,X	EP,A,0 454 636 (SGS-THOMSON MICROELECTRONICS S.R.L.) 30 October 1991	7,12,17, 18

Form PCT/ISA/210 (extra sheet) (January 1985)

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO. US 9205720 SA 62375

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information. 16/10/92

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
GB-A-2190521	18-11-87	US-A-	4811208	07-03-89
		DE-A-	3716229	19-11-87
		FR-A-	2598835	20-11-87
		JP-A-	62286128	12-12-87
EP-A-0213843	11-03-87	US-A-	4777588	11-10-88
		JP-A-	62052637	07-03-87
EP-A-0170284	05-02-86	JP-C-	1625134	18-11-91
		JP-B-	2048931	26-10-90
		JP-A-	61040650	26-02-86
		US-A-	5093783	03-03-92
US-A-4212076	08-07-80	DE-A,C	2743060	13-04-78
		GB-A-	1555609	14-11-79
		GB-A-	1557610	12-12-79
		JP-C-	1265431	27-05-85
		JP-A-	54048137	16-04-79
		JP-B-	59042892	18-10-84
		US-A-	4331893	25-05-82
EP-A-0454636	30-10-91	None		

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82